Extending Deep Learning Techniques to Communication Systems

Michael Patel
North Carolina State University
Department of Electrical and Computer Engineering
mrpatel5@ncsu.edu

***Abstract*—In the era of Big Data, machine learning and deep learning have become ubiquitous and elegant tools for processing data in many forms (images, text, audio, speech, etc.) In particular, deep learning has been transformative in the fields of Computer Vision (CV) and Natural Language Processing (NLP). However, there are other areas of science that may want to employ deep learning to handle large amounts of data. Communications systems have been designed and operated successfully for transmitting and receiving data for many years without deep learning techniques, but there may be opportunities to grow and improve.**

In this paper, I focus on the replacement of the traditional transmitter-channel-receiver model with an end-to-end neural network representation in order to minimize bit-error-rate (BER). While this approach is relatively unstudied, the field of communications may gain a more nuanced perspective.

This paper outlines some introductory and background material before describing my system model, implementation, and experimental results. Then, I critique aspects of the research paper I followed. Finally, I lay out some extensions of future work on the subject that may yet be explored in continuing research.

I. Introduction

Since data must be originated and be received somewhere, wireless communications and big data make for an interesting pair of companions. Mobile applications in smartphones, laptops, tablets, and the burgeoning Internet-of-Things (IOT) all take part in transmitting and receiving massive amounts of data at relatively fast speeds. Additionally, the popularity of these technologies will only make the challenge to satisfy user convenience more demanding. Consequently, modern communications technologies must be able to handle lots of data. Some methodologies such as 5G, massive MIMO, and mmWave have already gained significant traction [2].

However, there may be another approach to tackling the big data challenge. In general, deep learning has been most recognizably used as a tool for applications to process lots of data (images, speech, audio, etc.) Thus, it is of no coincidence that the emergence of successful deep learning arrived with the blossoming of big data.

Many times, traditional communication models assume linear, stationary, Gaussian, or deterministic components, which at best, can only approximate channel environments [1], [2]. Unfortunately, channels have imperfect realities, so even the best models today are limited. Deep learning can be used to rethink end-to-end reliable data transfer between transmitters and receivers. Therein lies the potential of applying deep learning techniques to the physical layer; the goals of communication systems may be achieved with deep learning tools.

II. Background

The inspiration for a neural network representation of the traditional transmitter-channel-receiver model is due to the ability for neural networks to process data in massive parallel using GPU hardware. The communication between many transmitters and receivers can be described as a feedforward network of signals with backpropagation error messages that eventually converges to a system with a robust channel medium.

A. Physical Layer

When considering the interdisciplinary aspects between signal processing and deep learning, it may be valuable to examine the intersection through the lens of computer networking. In computer networking, the OSI teaching model may be a good reference point. The OSI model is an abstraction of a network structure that is composed of several layers of protocols, technologies, and standardizations. Layer 5 of the OSI model is called the Application Layer. This is where deep learning provides obvious and immediate results as part of end applications (like Netflix recommendation systems).

Nevertheless, lower layers in the OSI model stack may also be able to take advantage of deep learning. Specifically, Layer 1, the Physical Layer, is the actual communication channel that has extensive demands to perform lots of signal processing (lots of data) at real-time speeds (wired or wirelessly) without consuming too much energy in the process. Here, deep learning may be able to augment performance and improve efficiency. This would directly tie together the fields of communications and data science.

B. Neural Networks

One of the defining distinctions between physical layer communications and deep learning is quite fundamental. Communications, particularly telecommunications, have a long history of performance built from expert domain knowledge, information theory, and statistics. In essence, signal processing utilizes deductive types of reasoning. Meanwhile, machine learning forgoes this and improves performance through experimentation and learned behaviours. Indeed, much machine

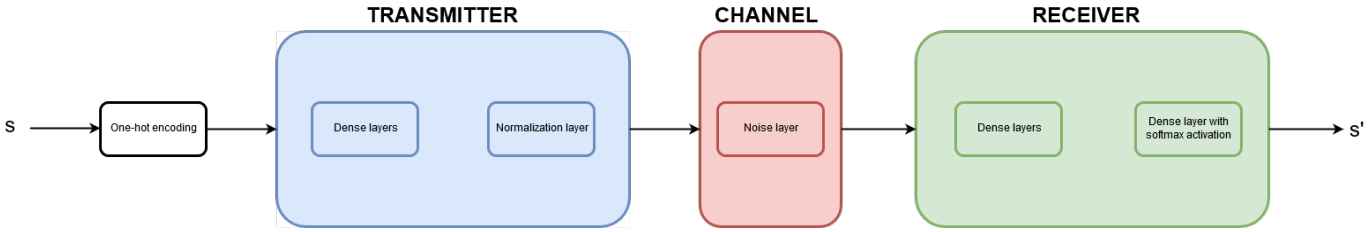

Fig. 1. Architecture presented by O'shea et. al.

learning research has been driven by heuristics and evidence-based observations.

Furthermore, the traditional transmitter-channel-receiver model is generally outlined as a series of function blocks that perform sequential tasks such as encoding, modulation, and filtering. Using expert domain knowledge, the individual function blocks can be optimized to produce better performance [1]. While the motivation to improve performance is the same in machine learning, it strives to accomplish this by optimizing the system as a whole. Instead of a per-block basis, optimizations are made end-to-end. This key difference may lead to important discoveries.

Before describing a neural network representation of the communications system model, I want to note that what makes a network deep is due to the neural network having depth via several hidden layers.

III. System Model

The communications system in the framework of deep learning can be examined as an autoencoder. An autoencoder is a type of neural network that is trained to attempt to copy inputs to outputs. This definition translates to the conventional transmitter-channel-receiver model which attempts to reconstruct a transmitted signal at the receiver error-free. The copying functionality is constrained, so the reconstruction is only approximated. Otherwise, the autoencoder simply learns to copy inputs to outputs which does not generalize well to different types of input data (this is a case of model overfitting).

The autoencoder learns features about the data in an effort to perform generative modelling that resembles the data. The autoencoder also does dimensionality reduction of the data (akin to data compression). At a high level, the autoencoder can be decomposed into two components: an encoder that maps the data to a lower dimensional space, and a decoder which reconstructs the data from that lower dimensional mapping. The purpose is to achieve this with a very small probability of error between the input and outputs of the system. Therefore, it is quite analogous to the goals of traditional communications systems.

The encoder is a function f that takes in as input data x to produce a lower dimensional representation h. In other words,

$$h = f(x)$$

The decoder is a function g that performs reconstruction on h to produce output r. In other words,

$$r = g(h)$$

Ideally, the errors between x and r occur with very small probabilities.

The encoder can be studied as a transmitter, while the decoder can be considered a receiver. The channel is represented by the lower dimensional representation h. After training, the autoencoder should be robust to typical channel distortions such as noise and fading. The autoencoder learns how to perform well even with the presence of background noise.

IV. Methodology

Following the work done in [1], the autoencoder architecture is shown in Figure 1. It is composed of a dense and normalization layer in the transmitter, a noise layer for the channel, and several more dense layers in the receiver. The input to the network is a one-hot encoded vector. I used this network structure as a guideline, but added more layer depth and regularization than initially described in [1]. I also modified the input to the network from using one-hot encoding vectors to using an embedding layer instead. Furthermore, I added a layer after the transmitter layers, but before the noise layer that performs Rayleigh fading. This layer in the architecture can be optionally included or excluded via command line arguments. Figure 2 shows a current model summary, and Figure 3 shows a current backend graph model. The model is single-input and single-output. In other words, the system contains only one transmitter and one receiver with no interfering users in between.

V. Implementation

A. Project Environment

Language: Python 3.6, Matlab
Libraries: TensorFlow, Keras, Numpy, Matplotlib, Pandas
Dataset: custom matrix of one-hot encoded row vectors
GPU: NVIDIA GTX 1070

B. Data Representation

A challenge with translating a communications model into a neural network representation has been how to best structure the input to the network. The network needs sets of training, validation, and testing data in the form of numerical tensors, but communications operates on signals. O'shea et. al. used a series of one-hot encoded vectors to represent a signal, but I have used a matrix where each row is itself a one-hot encoded vector. The column dimension is indirectly determined by the

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 16, 16)            256
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 16)                4112
_________________________________________________________________
batch_normalization (BatchNo (None, 16)                64
_________________________________________________________________
dense_1 (Dense)              (None, 16)                272
_________________________________________________________________
batch_normalization_1 (Batch (None, 16)                64
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 34
_________________________________________________________________
batch_normalization_2 (Batch (None, 2)                 8
_________________________________________________________________
lambda (Lambda)              (None, 2)                 0
_________________________________________________________________
gaussian_noise (GaussianNois (None, 2)                 0
_________________________________________________________________
dense_3 (Dense)              (None, 16)                48
_________________________________________________________________
batch_normalization_3 (Batch (None, 16)                64
_________________________________________________________________
dense_4 (Dense)              (None, 16)                272
_________________________________________________________________
batch_normalization_4 (Batch (None, 16)                64
_________________________________________________________________
dense_5 (Dense)              (None, 16)                272
=================================================================
Total params: 5,530
Trainable params: 5,398
Non-trainable params: 132
```

Fig. 2. Summary of my system model

number of bits k. From k bits, M different messages can be formed by $M = 2^k$. The column dimension of the matrix is M (the length of the row vector is M). The number of row vectors (number of rows) is given by the size of the training data set. The size of the training data set is arbitrarily $40,000$. The sizes of the validation and testing sets are likewise defined arbitrarily.

Determining how large to make each data subset has been a challenge itself. There is not much literature on the subject of how to best represent signals as tensors for neural network purposes (This topic should receive more attention as preprocessing and model feeding are often the time and memory bottlenecks in neural network projects). I experimented by creating a dataset of $55,000$ row vectors with $40,000$ for training, $5,000$ for validation, and $10,000$ for testing.

C. Hyperparameters and Architecture Choices

1) Loss Function: The loss function selected is categorical cross entropy because cross entropy is a useful measure for output described as a probability between 0 and 1. The generated outputs of the autoencoder are compared with the input values of the data set, so the error is binary, either true (no error) or false (error).

The cross entropy loss function is used in information theory to quantify the differences of two data distributions (usually through KL-Divergence). In the system model, the probability distributions belong to the true data distribution and the reconstructed data distribution. The difference measurement between the distributions represents the error that is minimized over training time.

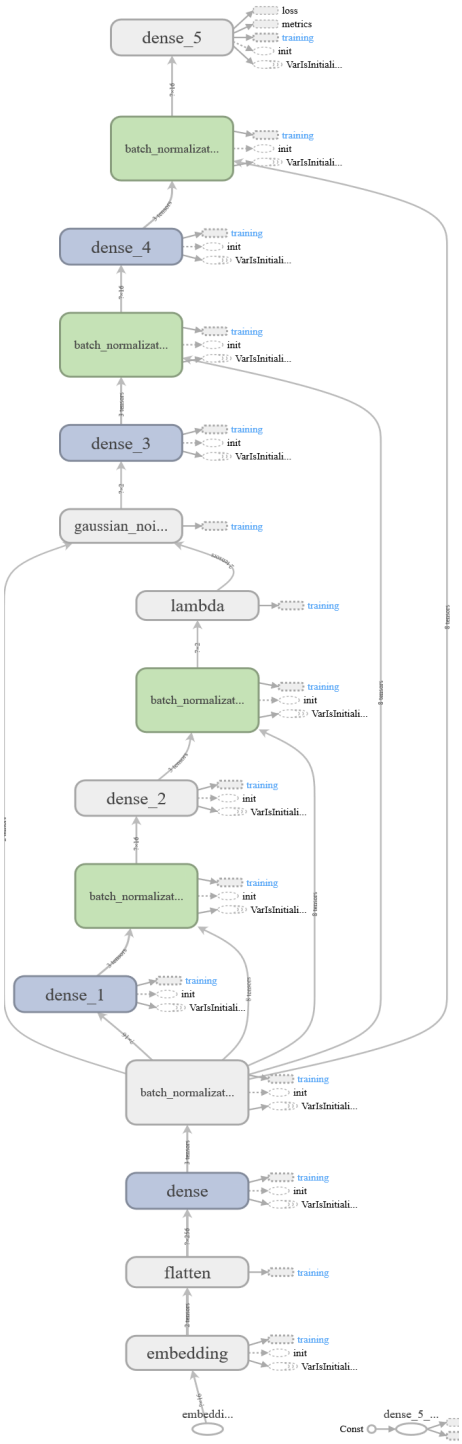

Fig. 3. Summary of my graph model

2) Optimizer: The optimizer chosen is Adam, which is frequently used in many generative models. The backend of Adam is quite memory friendly and has proven to reliably converge when the training data is sparse. I set the learning rate to 0.001 which provided decent training accuracy versus time tradeoffs. I also experimented with a lower learning rate of 0.0001.

3) Activations: For many of the dense layers, I am using relu activation functions to introduce non-linearities and a final softmax activation layer to produce the multi-class output which is the same as [1]. Softmax activation is generally used in pairing with a cross entropy type of loss function. The outputs of the softmax activation illustrate a probability distribution over the predicted row vector (i.e. a probability for each particular value of M in each reconstructed signal).

4) Number of Epochs: I trained the system model neural network for 12 epochs before noticing signs of overfitting when using noise and Rayleigh fading channel layers. During the experiment phases of training, I swept through a varying amount of epochs to use from 5 to 30 in increments of 1. The overfitting and relatively quick training process is related to how the signals are represented as input data. The matrix is sparse, so the network can perform computations easily. To combat the overfitting, I reduce the number of training epochs and batch size. To combat the slower training time, I used batch normalization layers.

5) Batch Size: The selected batch size hyperparameter was 16. During experimentation, a batch size greater than 32 offered poor BER results. The smaller batch size had lower BER, but took more time to train per epoch. I swept through a varying batch size from $[16, 32, 64, 128]$.

A final training aspect is with regards to communications choices. O'shea et. al. set the SNR to $7dB$ while performing training, but this is ambiguous as to whether this is the optimal value for which to conduct training. Furthermore, the autoencoder uses a noise layer with a variance equal to $\frac{1}{2*R*\frac{Eb}{N_0}}$ where R is the data rate and $\frac{Eb}{N_0}$ is the energy per bit to noise power ratio. The construction of the noise is yet another design choice to explore.

VI. Results

Analysis of the autoencoder system model included BER curves as was done in the O'shea paper. The test data set is fed forward through the autoencoder. AWGN is then compounded along with an optional Rayleigh fading before comparing the input with the output for errors. The average error is computed and plotted against a range of SNR (in dB). The BER curves plotted the SNR values from -15 dB to 15 dB at 1 dB increments.

A. Training SNR

The training SNR value was referenced as 7 dB in [1]. I conducted model training at 0, 7, and 15 dB. Consequently, the training SNR was treated as a training hyperparameter. Figures 4, 5, and 6 show the BER curves for different training SNR values. There is not much difference in BER curves between

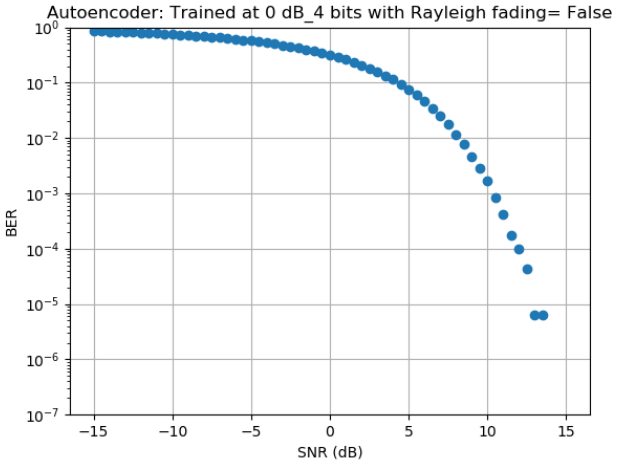

Fig. 4. Trained at 0 dB

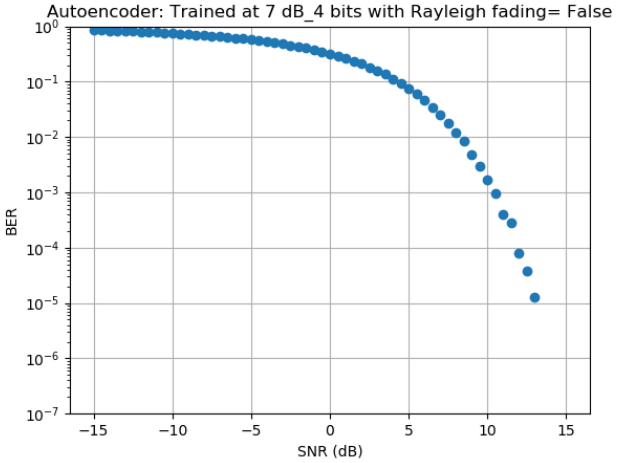

Fig. 5. Trained at 7 dB

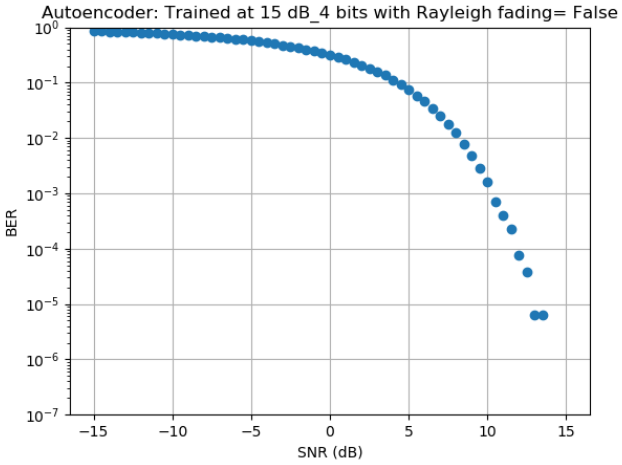

Fig. 6. Trained at 15 dB

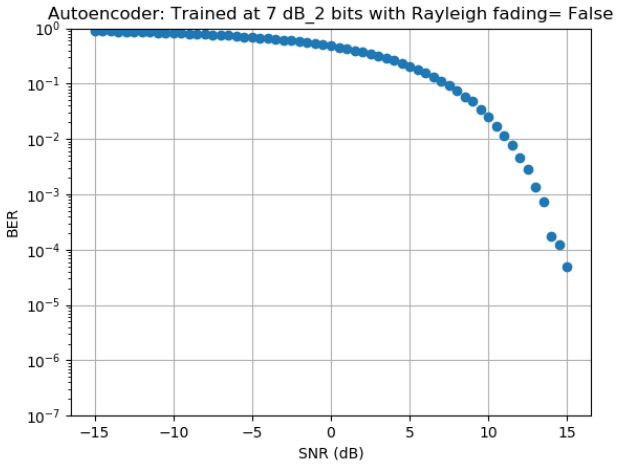

Fig. 7. Use k = 2 bits

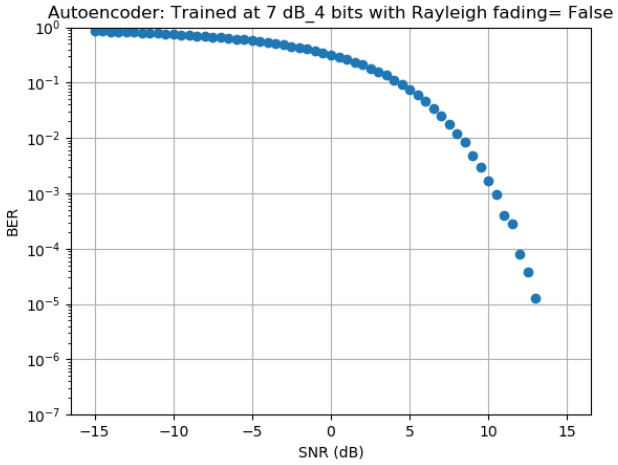

Fig. 8. Use k = 4 bits

the three BER curves, if holding the number of bits constant. As [1] has detailed, choosing an SNR value at which to train the autoencoder is not obviously intuitive. Upon first observation, it may seem that the model learns the communications system appropriately regardless of SNR. However, this should be taken as a cautionary conclusion that should prompt an invitation for more research and investigation.

B. Data Rate

The data rate $R = \frac{k}{n}$ where k is the number of bits and n is the number of discrete uses of the channel. The number of possible messages $M = 2^k$. Increasing the number of bits k (and therefore the data rate R) did lead to better performance at the same fixed SNR training value. While higher values of k and M are desirable, the tradeoff is computational complexity.

The data representations of the input signals are matrices that grow with M. Too high of an M value can quickly exhaust precious physical memory resources. If this happens, then the model has suffered from "the curse of dimensionality". Figures 7 and 8 show the BER curves at varying k values. The higher k values show a steeper curve and better BER at the same SNR level. For example, at 10 dB, the BER value is 10^{-2} for $k = 2$ and 10^{-3} for $k = 4$. At higher k, better performance is achieved.

C. Rayleigh Fading

While the O'shea paper focused on an AWGN channel, I also experimented by modifying the system model with an additional layer that performs Rayleigh fading. This was done in an effort to make the channel more robust for final testing. The use of Rayleigh fading is meant to mimic the transmission to the receiver that does not include a line-of-sight. This extension from the O'shea paper offered an interesting study path for implementing deep learning in communications as I did not encounter any other research that also explored Rayleigh fading with an autoencoder architecture.

The structure of the neural network is optionally defined through command line arguments. By default, the architecture only includes an AWGN channel. By specifying the rayleigh argument option, the network will include the insertion of a layer that conducts Rayleigh fading after the transmitter block, but before the noise layer.

The rayleigh argument option also specifies the generated BER curve when performing model evaluation on the test data set. Figures 9 and 10 show BER with and without the Rayleigh fading. AWGN is always included. With Rayleigh fading, BER performance is worse, evidenced by the curve becoming less steep even at higher SNR values.

D. Comparison with O'shea paper

The results of my system model illustrate that I do not achieve the same BER performance as shown in [1]. O'shea et. al. define an autoencoder with $k = 4$ and $n = 7$. Figure 11 shows the BER curve from my system model for the same k and n values. In the O'shea paper, the BER curve begins to steep at approximately 2 dB. At 4 dB, the error is roughly 10^{-2}; at 6 dB, the error is roughly 10^{-3}; and at 8 dB, the error is roughly 10^{-5}.

My own model never reaches 10^{-3} even at SNR values as high as 15 dB. My BER curve is clearly not as steep as the same curve presented in the O'shea paper. While not much is disclosed about how the authors achieved that particular output, my speculation is either they calculated the errors differently than myself, or they structured their data input differently through some advanced form of preprocessing.

E. Comparison with BPSK and QAM

The last area of study was to compare model performance against standard digital modulation schemes such as BPSK and QAM. Channels with simulated AWGN and AWGN plus Rayleigh fading were plotted in Matlab software. The BER curves are shown in Figures 12 and 13. Interestingly, at low SNR, the autoencoder model performs quite poorly against 16-BPSK and 16-QAM. At approximately 5 dB, the autoencoder

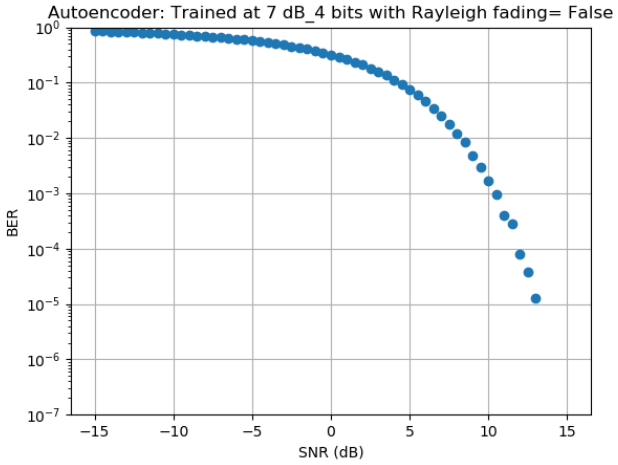

Fig. 9. No Rayleigh fading

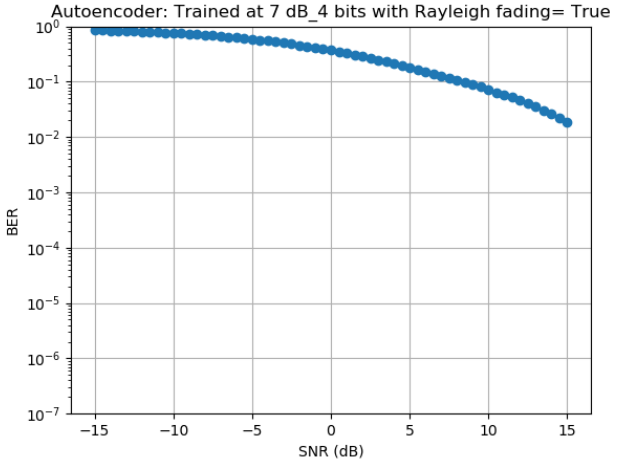

Fig. 10. With Rayleigh fading

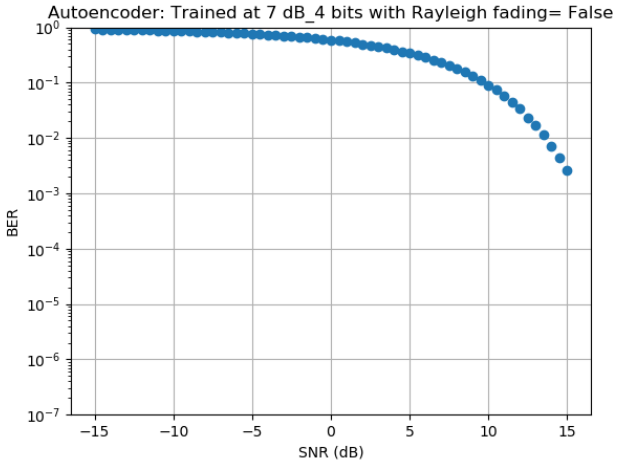

Fig. 11. With k=4 and n=7 to compare with [1]

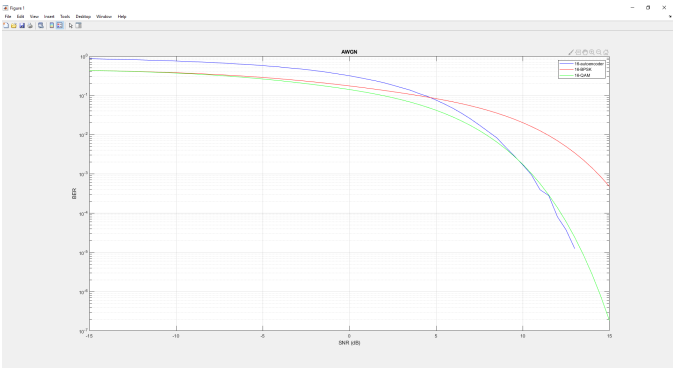

Fig. 12. Comparison with AWGN channel

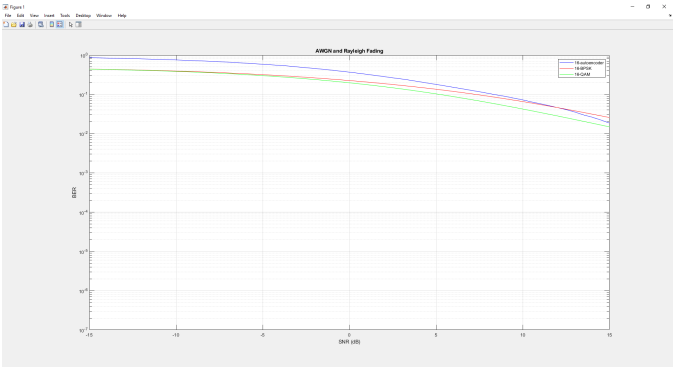

Fig. 13. Comparison with AWGN and Rayleigh fading channel

steeps quickly, outperforming BPSK. From 9 to 15 dB, the autoencoder BER curve is similar to QAM, doing slightly better at the very end.

Similar observations can be made about the autoencoder against BPSK and QAM with Rayleigh fading. At SNR values below 10 dB, the autoencoder is no match for BPSK and QAM. However, soon after, it outperforms BPSK and trends towards QAM.

This bit of visualization uniquely captures the spirit of this investigative foray into deep learning from the communications perspective. With a relatively simple neural network structure, decent BER performance can be modelled, even against some popular modulation techniques.

VII. Critique of O'Shea Paper

My discussion about deep learning in the context of communications concludes with an honest critique of the research presented in [1]. The paper does an excellent job of detailing their neural network structures and output dimensions per layer. This became exceedingly useful when constructing my own autoencoder. Even though the authors do not share their code, they describe enough about the architecture that I was able to infer the layer structure and write my own code. Specifically, Table IV in [1] outlines the basic autoencoder model that I used as my initial baseline before modifying with Rayleigh fading and added network depth.

The paper was light on the literature survey section. There were not many references to other research that also involves deep learning and communications. This made it difficult to find other reading resources that would have aided in my own learning on the subject. Understandably, this may have been because the pool of publications is shallow.

A technical challenge I experienced was how to best represent the input data signal as I fed it into the autoencoder model for training. This was mentioned in the paper, but not with much guidance. This step is crucial as typical bottlenecks in deep learning include data preprocessing and model feeding. While I used a matrix of one-hot encoded row vectors and an initial Embedding layer, I do feel that there may be a better methodology to represent the data. One alternative may be to use raw data captures from either the field or Matlab simulations. Then, preprocessing can be done before feeding to produce the lower-dimensional embedding.

Another useful aspect would be with regards to how the authors generated their BER curves from the outputs of their autoencoder models. This would help to directly compare the results of my system model with those presented in the paper.

VIII. Conclusion and Future Work

Despite some skepticism about the implementation of the autoencoder approach, the use of deep learning in communications is exciting and promising. The preliminary results described in this paper's discussion highlight some key advantages for model prototyping and interesting avenues in which to conduct future research. Although the model analysis was limited to AWGN channels, Rayleigh fading, and single-input-single-output, the autoencoder performance against BPSK and QAM made a good case for why deep learning should be studied in the context of communications systems.

Because of huge demands for data transmission, future communications systems will have to adapt and meet the challenge. Deep learning is a tool that can be used to conquer the challenge possibly in step with parallel developments in 5G and massive MIMO. Applications of the future will consume and produce data at such high rates that the data processing must be handled with greater care, privacy, sensitivity, and intelligence. Big data presents many grand challenges that are still open and unresolved.

Extensions of the work presented in this paper and [1] would include focusing on input data representation, multiple interfering users, and channel construction. AWGN and Rayleigh fading are only the tips of the iceberg. Perhaps deep learning should be further explored in conjunction with 5G and massive MIMO for learning more nuanced insights about channel states and channel information. Deep learning and modern communications are both exciting fields brimming with innovative research that may have more to gain by working together.

References

[1] T.J. OShea and J. Hoydis. (2017) An introduction to deep learning for the physical layer. [Online]. Available: https://arxiv.org/pdf/1702.00832.pdf

[2] T. Wang, C.-K. Wen, H. Wang, F. Gao, T. Jiang, and S. Jin, Deep learning for wireless physical layer: Opportunities and challenges, China Communications, vol. 14, no. 11, pp. 92111, 2017.

Appendix A
Copy of Paper